Embedded Linux Conference Europe

Is Linux Kernel Development Good Enough to Make Your Life Depend on it?

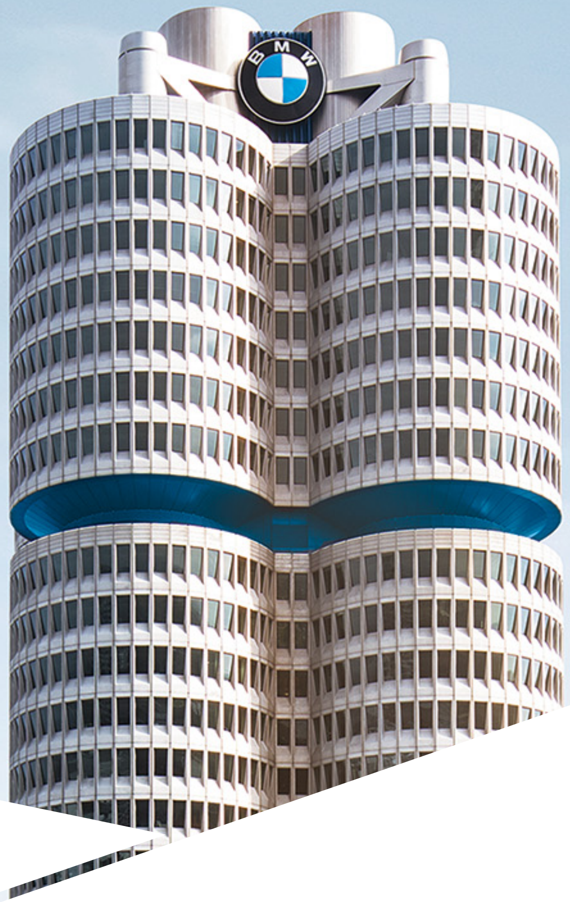
Progress on Procedures & Methods to Qualify the Linux Kernel Development Process

Lukas Bulwahn

THE LINUX FOUNDATION

IS LINUX KERNEL DEVELOPMENT GOOD ENOUGH TO MAKE YOUR LIFE DEPEND ON IT?

PROGRESS ON PROCEDURES & METHODS TO QUALIFY THE LINUX KERNEL DEVELOPMENT PROCESS.



Lukas Bulwahn | October 24th, 2017

BMW GROUP

THE NEXT 100 YEARS

BMW Car IT

ABOUT BMW CAR IT GMBH

- Founded in 2001 as a wholly owned subsidiary of the BMW AG
- Strengthen BMW's software competence
  - View vehicles as software systems
  - Develop innovative software for future BMW Group vehicles
  - Prototype solutions for early and reliable project decisions
- Participate in several open-source communities and research projects

BMW Car IT GmbH is currently hiring!

http://www.bmw-carit.de/opportunities/

THE OSADL SIL2LINUXMP PROJECT

- **Mission:**
  - **Provide procedures and methods** to qualify Linux on a multi-core embedded platform at safety integrity level 2 (SIL2) according to IEC 61508 Ed 2.
  - **Show feasibility** of procedures and methods on a real-world system
  - **Show potential** for re-use of Linux kernel analysis

THE OSADL SIL2LINUXMP PROJECT

- **Mission:**
  - **Provide procedures and methods** to qualify Linux on a multi-core embedded platform at safety integrity level 2 (SIL2) according to IEC 61508 Ed 2.
  - **Show feasibility** of procedures and methods on a real-world system
  - **Show potential** for re-use of Linux kernel analysis

- **Collaborative project of industrial & research partners**
  - project running since 2015, organized by OSADL
  - Full members: BMW Car IT, Intel (since ‘17), A&R Tech, KUKA, Sensor-Technik Wiedemann (full members till ’16, reviewing members in ‘17)
  - Reviewing members: Bosch, Elektrobit, Hitachi, Linutronix, MBDA Italia, MEN Mikro Elektronik, Mentor, OpenSynergy, Pilz GmbH & Co. KG, Renesas, Vienna Water Monitoring Solutions
  - Academic members: Alexey Khoroshilov (ISP RAS), Kinggo Chow (Lanzhou Univ.), Julia Lawall (Inria/LIP6), Frank Tränkle (HS Heilbronn)
  - Experts from certification bodies: Bernhard Nölte (TÜV Süd), Oliver Busa, Robert Heinen, Hendrik Schäbe (TÜV Rheinland)
  - SIL2LinuxMP core working team: Nicholas McGuire, Andreas Platschek, Lucas Böhm, Markus Kreidl (OpenTech)

OVERVIEW

- **Setting the Scene**
  - Introduction to Functional Safety
  - Assumed System Architecture
- **Linux Safety Qualification**
  - Safety Assessment of Pre-existing Software
  - Hazard-driven Decomposition, Design & Development
  - Functional-driven Selection versus Assurance-driven Selection
- **Linux Kernel Development Quality Assessment, Assurance and Improvements**
  - Methods & Tools for Analysis of the Linux Kernel Development
  - Improvements in the Linux Kernel
- **Conclusion**

SETTING THE SCENE

FUNCTIONAL SAFETY

“**Functional safety** is the part of the overall safety of a system (…) that depends on the system (…) **operating correctly in response to its inputs**, including the safe management of likely **operator errors, hardware failures and environmental changes**.

The **objective of functional safety** is freedom from **unacceptable risk** of physical injury or of damage to the health of people either directly or indirectly.”

(Source: wikipedia.org:Functional Safety)

FUNCTIONAL SAFETY

“**Functional safety** is the part of the overall safety of a system (…) that depends on the system (…) **operating correctly in response to its inputs**, including the safe management of likely **operator errors, hardware failures and environmental changes**.

The **objective of functional safety** is freedom from **unacceptable risk** of physical injury or of damage to the health of people either directly or indirectly.”

(Source: wikipedia.org:Functional Safety)

Work on Functional Safety is **Risk Management**

- Risk Management is to **focus quality assurance on the right aspects and right parts!**
- It is NOT to do just more work or write hundreds of documents!

FUNCTIONAL SAFETY

- How to determine the **acceptable risk?**
  - Agreement in **global safety standards**
    - **IEC 61508**: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
      - a basic functional safety standard applicable to all kinds of industry
      - Adaptations to specific industries: ISO 26262 for automotive, IEC 62279 for railway applications

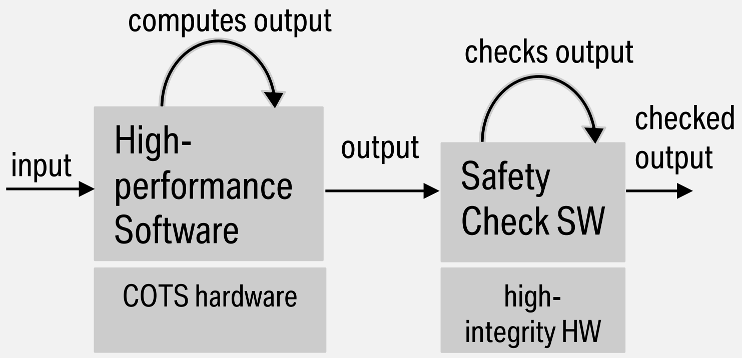
FUNCTIONAL SAFETY

- How to determine the **acceptable risk?**
  - Agreement in **global safety standards**
    - **IEC 61508**: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
      - a basic functional safety standard applicable to all kinds of industry
      - Adaptations to specific industries: ISO 26262 for automotive, IEC 62279 for railway applications

- How to **design safe systems?**
  - **System design & system analysis**
    - Analyze your system to know which parts must be of high quality for the system's safety
      - Assign safety integrity levels (SIL) to those parts, SIL1 (low safety level) to SIL4 (high safety level)
  - **Rigorous development process**
    - Develop those parts with high SIL with sufficient rigor (= the right development process)
      - Safety standards state which objectives shall be achieved in each development phase

ALTERNATIVE SYSTEM ARCHITECTURES

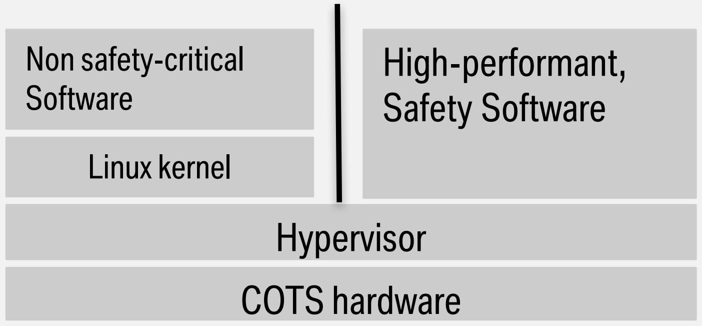
System Architecture A



Assumptions/Drawbacks:

- Needs decomposition of safety of complex function to a simple checking on lower performance high-integrity HW
- Checking must detect subtle errors from HW and OS of high-performance computation.

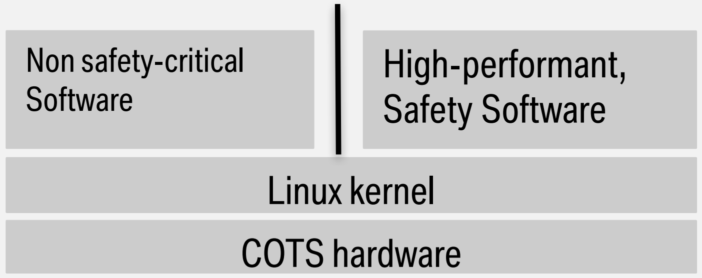
System Architecture B



Assumptions/Drawbacks:

- Hypervisor & HW guarantees isolation
- Safety SW without underlying OS
- If needed, scheduling, multi threading & file system is implemented in safety SW

System Architecture C



Assumptions/Drawbacks:

- Linux kernel provides sufficient isolation
- Safety SW uses Linux scheduling, multi threading, file system etc.
- Parts of Linux kernel and HW functionality must be qualified.

SIL2LinuxMP Project focusses on work for system architecture C.

LINUX SAFETY QUALIFICATION

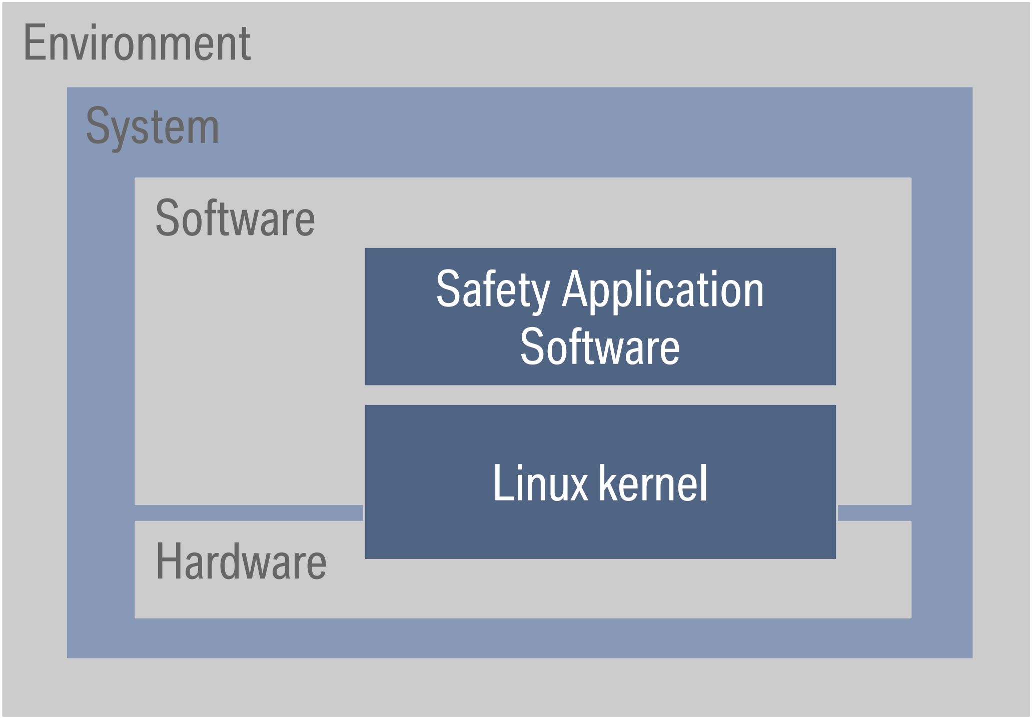
NOTABLE FACTS ON THE LINUX KERNEL

- **Linux is a large software project with an impressive change rate**
  - 23 million lines of code & 2 million lines of documentation
  - 14,000 commits in every release/every 70 days
  - 17,000 developers in total history, 1,700 developers in each release
  - kernel developers are affiliated with many different companies or act as individuals
- **Development process**
  - Highly transparent due to open-source character
  - Process defined and enforced by social contract, but not legal working contracts
- **Stabilizing phase of Linux LTS kernel versions**
  - 4036 bug-fix commits in the 4.9 branch until v4.9.55 => ~90 bugs corrected each week
  - detected by continuous developer review, various verification activities and execution on billion devices

STARTING THE SAFETY QUALIFICATION OF THE LINUX KERNEL

How can the Linux kernel cause physical injury or damage to the health of people?

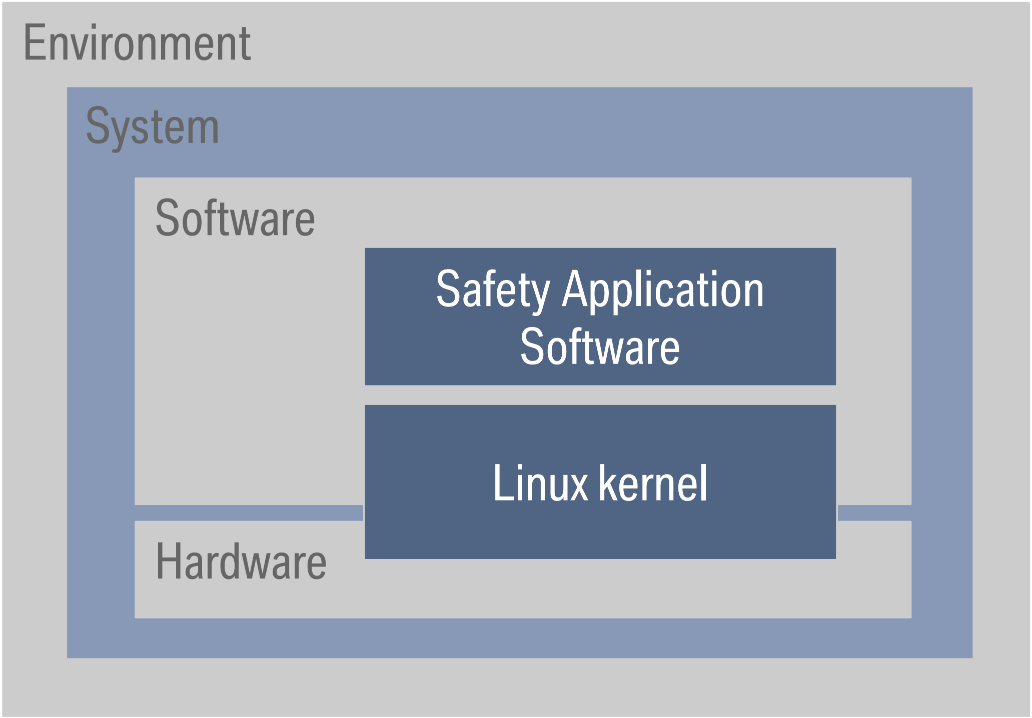
- It depends on Environment, System, Hardware and Safety Application Software
- Providing a generic answer for all potential systems:
  - would make a large number of system assumptions,
  - and system assumptions could not be implementable or unrealistic for a specific system design.



STARTING THE SAFETY QUALIFICATION OF THE LINUX KERNEL

How can the Linux kernel cause physical injury or damage to the health of people?

- It depends on Environment, System, Hardware and Safety Application Software
- Providing a generic answer for all potential systems:
  - would make a large number of system assumptions,
  - and system assumptions could not be implementable or unrealistic for a specific system design.



Conclusion: Understand & Use the System Context!

- SIL2LinuxMP chose a simple example system to understand the activity, to be repeated for each system.
- Linux qualification is always done for each specific system.

Don’t claim Linux was used in a safety-critical system before, so your system will be safe without further consideration.

SAFETY ASSESSMENT OF PRE-EXISTING SOFTWARE

Problem for safety assessment of pre-existing software:

"Software was **not developed with safety** in mind" or "Software was **not developed with this system** in mind"

– Software development is already **done with a fixed process**.
– Specific **system context was not considered** in software development.

SAFETY ASSESSMENT OF PRE-EXISTING SOFTWARE

Problem for safety assessment of pre-existing software:

"Software was **not developed with safety** in mind" or "Software was **not developed with this system** in mind"

– Software development is already **done with a fixed process**.
– Specific **system context was not considered** in software development.

Solution to safety assessment of pre-existing software:

– Show assumed software context fits to specific system context
  – **System-specific** activity: Determines which functionality of the Linux kernel shall be assessed
– Assess development process and show compliance to safety standard
  – **Development-process-specific** activity: Determines if development is done with sufficient rigor and mitigates gaps

SAFETY ASSESSMENT OF AN OPERATING SYSTEM

- **Operating System:**
  - provides significant context-unspecific functionality
  - has a large hardware-software interface
  - potentially all functionalities can impact safety
- **System Design Goal:**
  - make system safety depend on a few selected OS functionalities
  - select OS functionalities your system relies on depending on the available assurances
- **Methods:**
  - Hazard-driven decomposition, design and development
  - Assurance-driven Selection

HAZARD-DRIVEN DECOMPOSITION, DESIGN & DEVELOPMENT (HD^3)

– **Goal** of the system analysis: **Precise technical safety requirements** on lower levels with traceability on system's safety impact

– **First naive system safety engineering**
  "The syscall open() is used in a safety-critical application and must work correctly."
  – Problem: Too imprecise to guide further testing, verification and validation activities

HAZARD-DRIVEN DECOMPOSITION, DESIGN & DEVELOPMENT (HD³)

– **Goal** of the system analysis: **Precise technical safety requirements** on lower levels with traceability on system's safety impact

– **First naive system safety engineering**
  "The syscall open() is used in a safety-critical application and must work correctly."
  – Problem: Too imprecise to guide further testing, verification and validation activities

– **Safety engineering with Hazard-driven Decomposition, Design & Development (HD³)**
  – Dedicated method to achieve more precision on complex systems with multiple levels for fault avoidance & detection
  – Results in 12 constraints on syscall open()!
  – Specific testing and verifications under those specific conditions is now feasible

System Call: open()

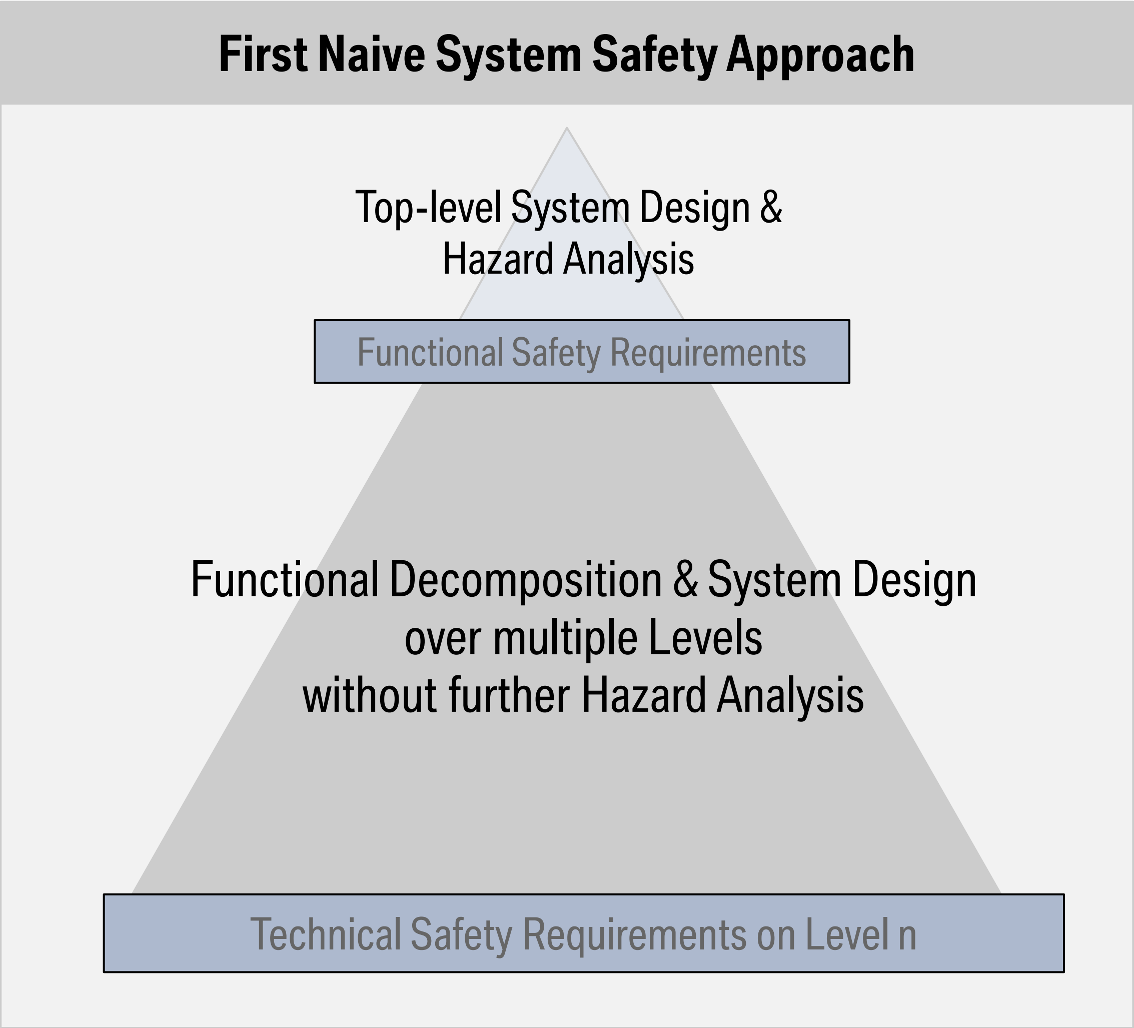
Origin: I/O Setup (ST_3020-ST_3029), Create File (ST_3040-ST_3049)

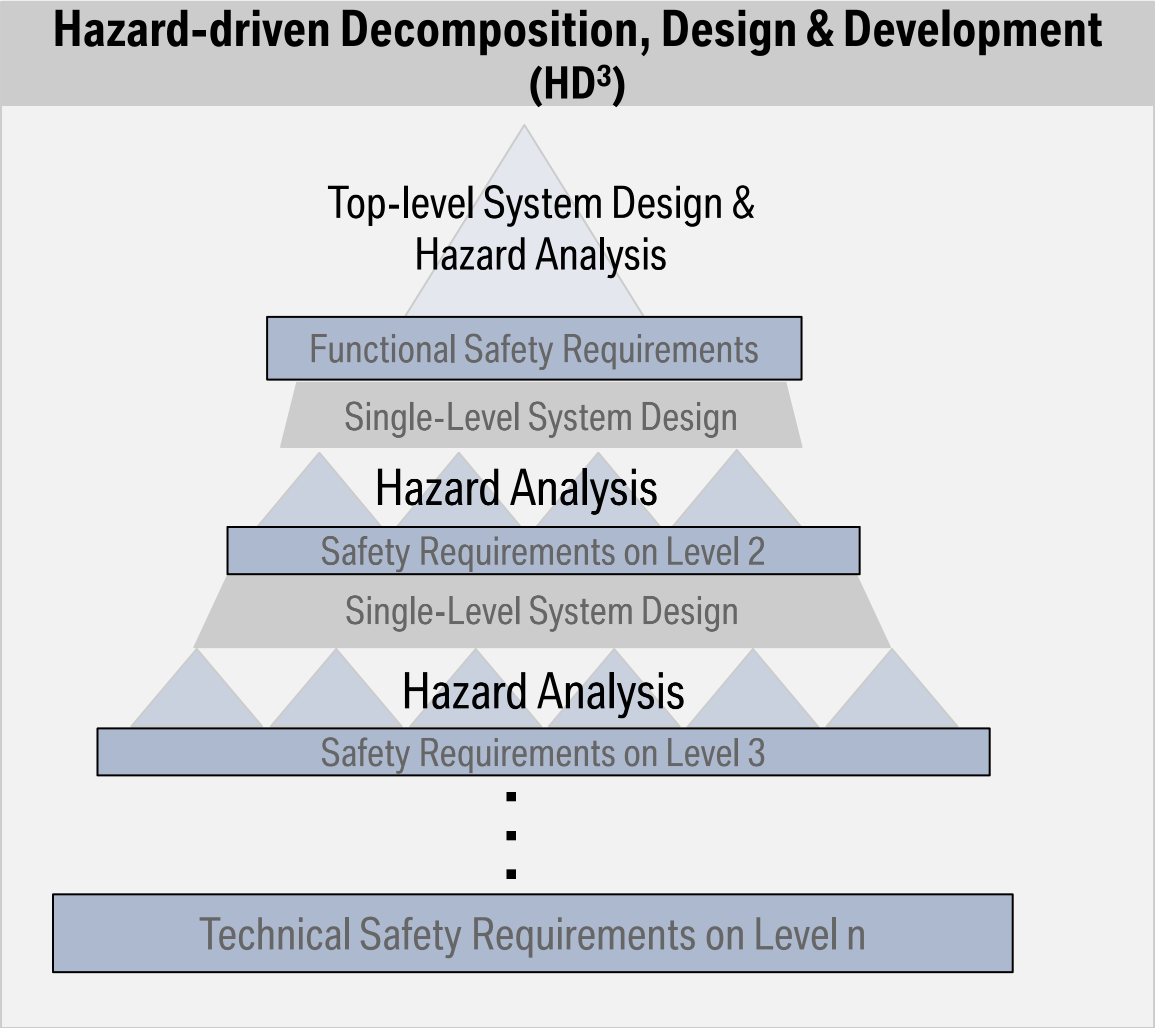
HAZOP IDs: SD_3520-SD_3529

References: 1) man 2 open 2) POSIX 1003.13 summary in section 6.6.1.4 3) open()

Related SACs: STS_3013 (minimize access priviledges), STS_3014 (access modes pre-determined), STS_3015 (init opens/creates files), STS_3019 (restricted usage of lseek()), STS_3028 (defined file creation flags), STS_3029 (check new files existence), STS_3030 (file creation logged), STS_3034 (file creation in non-RT), SDS_3508 (magic numbers prohibited), SDS_3509 (write files exclusively), SDS_3510 (set and verify permissions), SDS_3512 (file namespace specified), SDS_3513 (random bits in file names)

HAZARD-DRIVEN DECOMPOSITION, DESIGN & DEVELOPMENT





FUNCTIONAL-DRIVEN SELECTION VERSUS ASSURANCE-DRIVEN SELECTION

- **Example:** Implement an init system, that
  - sets up partitions for the applications with isolation and controlled access to shared system resources
  - starts up the safety applications in these partitions

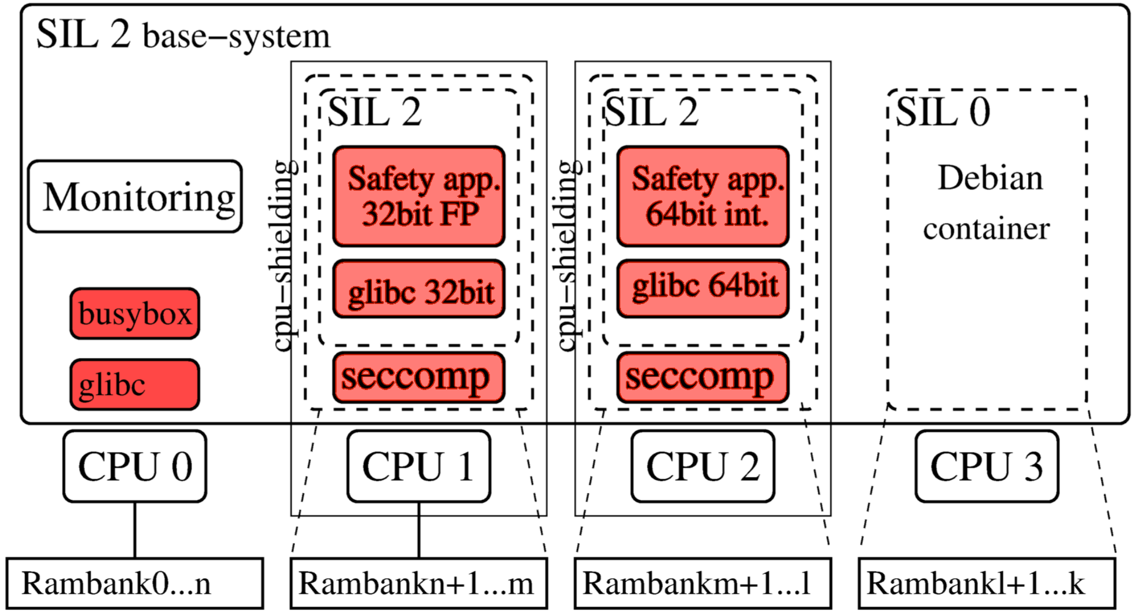
- Solution with **Functional-driven Selection**
  - Consider pre-existing technical solutions, init, systemd, etc.
  - Use systemd and write a few systemd service files

- Solution with **Assurance-driven Selection**
  - Consider technical solutions and determine potential and effort to qualify:
    - Special-purpose dedicated program using C libraries (libc, libseccomp, …)
    - pre-existing solutions, init, systemd etc.
  - Implement special-purpose dedicated program because:
    - sufficient evidences on quality assurance are not available for initd and systemd.

RESULTING SOFTWARE ARCHITECTURE

Main Architecture Decisions:

- Safety-critical and non-safety critical applications run on the same kernel
- Isolation is achieved with:
    - **CPU shielding**
    - use of dedicated cores and memory regions
- Unintended behavior of safety-critical applications is limited with **seccomp**
- System and applications are monitored with an application on a dedicated core
- Safety-critical applications use **glibc**



Source: Nicholas McGuire, SIL2LinuxMP High-Level Architecture, 2017.

LINUX KERNEL DEVELOPMENT QUALITY ASSESSMENT, ASSURANCE AND IMPROVEMENTS

NEW METHODS & TOOLS FOR ANALYSIS OF THE LINUX KERNEL DEVELOPMENT

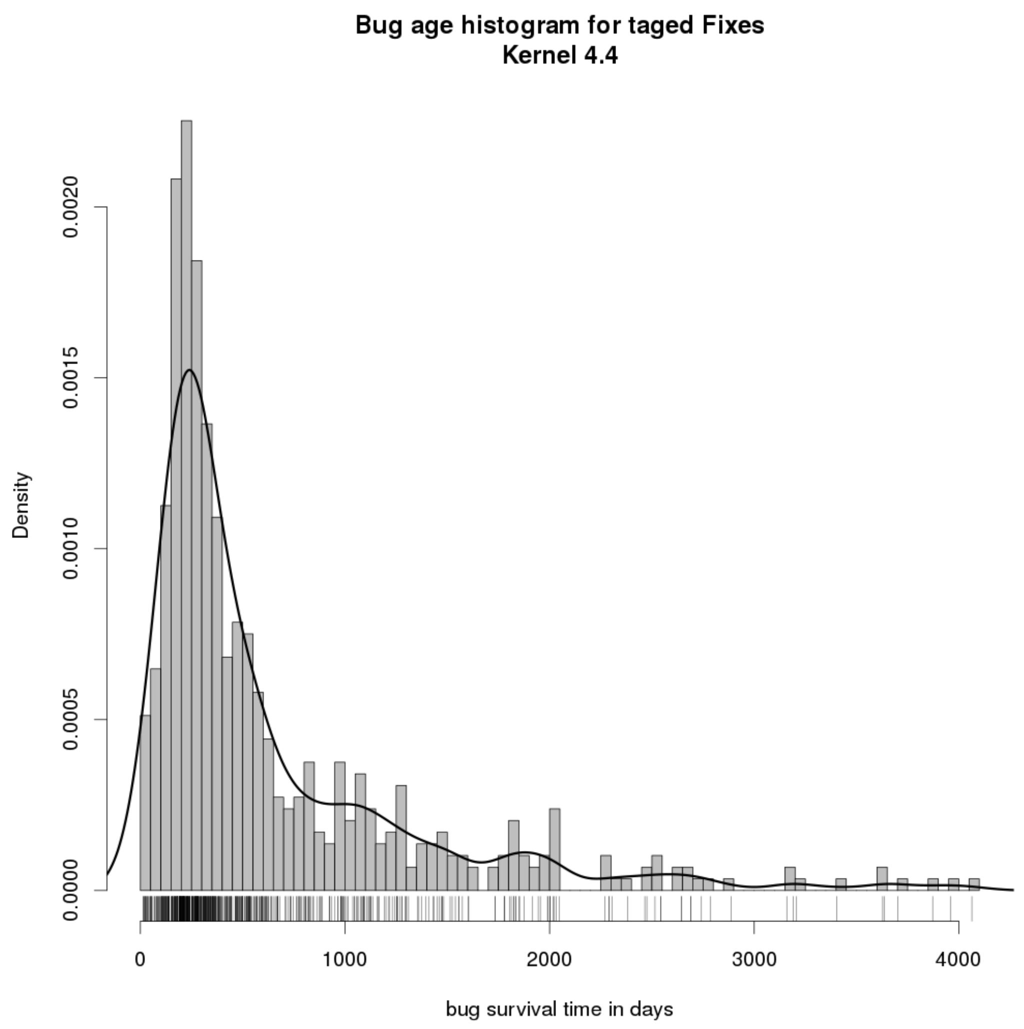
- **Analysis of kernel git meta data**
  - **Statistical prediction models** for the number of remaining bugs in the kernel
  - **Development life-cycle data mining**
    - **Quantify competence** of persons involved based on their activities
    - **Quantify dependencies** amongst developers and independence of persons doing code reviews
    - **Identify critical patches** that did get less review

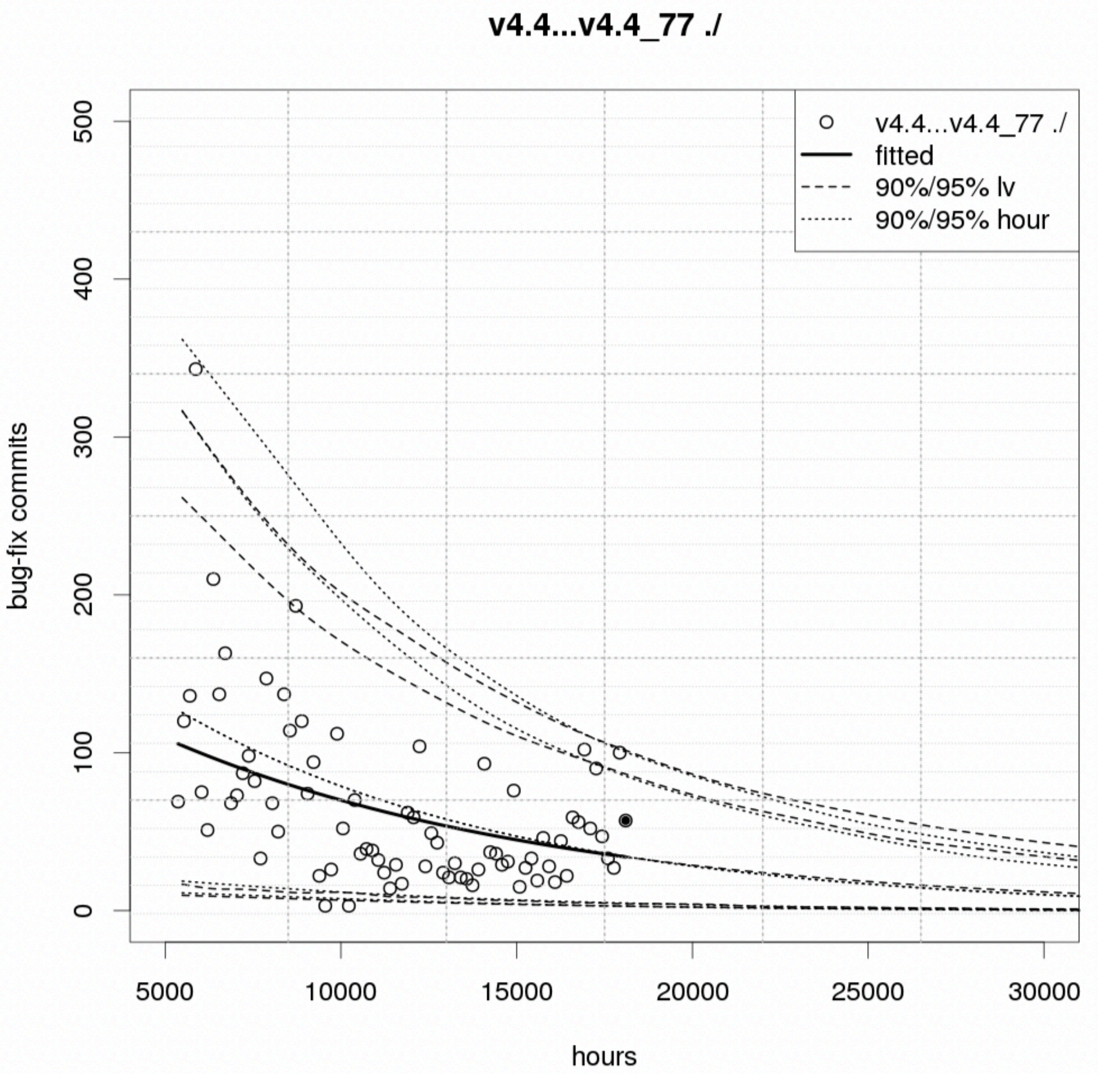
NEW METHODS & TOOLS FOR ANALYSIS OF THE LINUX KERNEL DEVELOPMENT

- **Analysis of kernel git meta data**
  - **Statistical prediction models** for the number of remaining bugs in the kernel
  - **Development life-cycle data mining**
    - **Quantify competence** of persons involved based on their activities
    - **Quantify dependencies** amongst developers and independence of persons doing code reviews
    - **Identify critical patches** that did get less review

- **Analysis of kernel source code**
  - **DB4SIL2**: Collect execution traces to argue independence of protection layers, path coverage, independence of consecutive calls and inherent diversity of system call executions
  - **Patch impact tester**: Determine if patches have impact on a specific kernel configuration
  - **Code minimization tool**: Preprocess selectively #ifdef & #if to minimize source code to relevant parts for review, inspection and source code analysis tools

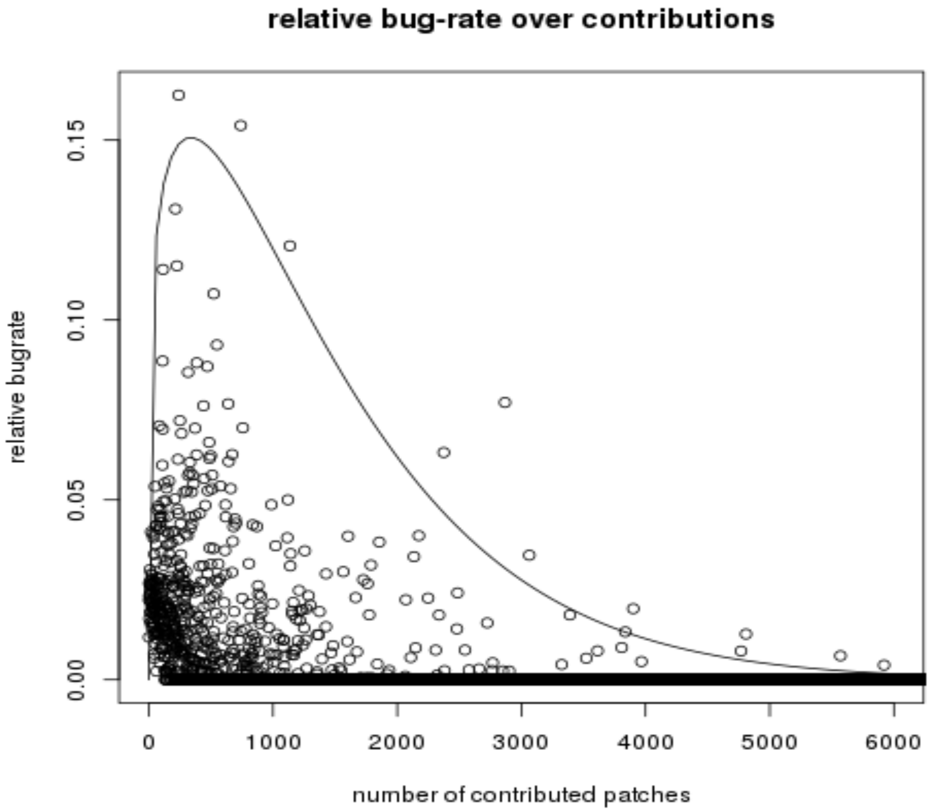
STATISTICAL PREDICTION MODELS





Source: Nicholas McGuire. SIL2LinuxMP Project. Identifying Stable Kernels - top-down model. August 2017.

CORRELATION BETWEEN BUG RATE AND NUMBER OF CONTRIBUTIONS



- Diagram shows the developers' bug rate average against the developers' total number of commits and its interpolation.
- So, what we learn from this?
  - Nothing really immediately applicable!
  - Some developers are significant bug producers (and the process does not stop them)
  - Consider yourself after 150 commits still as bloody beginner ;) Now, you will start creating bugs until you passed the 2000 commits…
  - Reviewers trust those developers with 100 to 1000 commits more than they actually deserve it
  - Get past the 3000 commits to avoid bugs ;)

Source: Nicholas McGuire, Andreas Platschek. Results from development life-cycle data mining. 2017.

IMPROVING THE LINUX KERNEL

- **Core observation**
  - Linux kernel developers know Linux in total better than anyone else, i.e., any internal team
  - Modifying Linux without following its development process reduces quality and increases risk of safety-critical bugs
  - To reduce risk, take a stable main-line Linux
  - **There is no Linux kernel for safety, it is just a well-matured LTS kernel**

IMPROVING THE LINUX KERNEL

- **Core observation**
  - Linux kernel developers know Linux in total better than anyone else, i.e., any internal team
  - Modifying Linux without following its development process reduces quality and increases risk of safety-critical bugs
  - To reduce risk, take a stable main-line Linux
  - **There is no Linux kernel for safety, it is just a well-matured LTS kernel**

- **Improving Linux**
  - Changes for improvements must follow the Linux development process
    - Improvements must be reviewed, accepted and appreciated by kernel developers
    - Engagement and interaction with on-going development is more effective than in a deferred post-development mode
  - Companies with Linux in safety-critical products should collaborate on quality assurance activities for Linux closely to kernel developers

NEEDED QUALITY ASSURANCE ACTIVITIES FOR SAFETY-CRITICAL SYSTEMS

- **Ongoing and future quality assurance activities**
  - **Coding style**
    - Respect existing coding style
    - Provide evidences for its quality
    - Monitor and motivate its compliance
  - **Testing**
    - Extend tests of the Linux Test Project for the determined syscalls
  - **Static analysis**
    - Detect more bug patterns and bug classes with coccinelle, sparse et al.
  - **Change management**
    - Track if bug fixes from main line are consequently backported
    - Analyze which kernel bugs & fixes impact the systems' safety
- Activities focus on parts of the Linux kernel relevant for the systems' safety

MAINTENANCE ACTIVITY: ROOT CAUSE ANALYSIS

- **Rationale** for doing root-cause analysis during operations
  - Safety standard requires continuous monitoring and analysis of identified issues
- **Implementation** with the Linux kernel development
  - Bugs are continuously found in Linux
  - Bug-fix commits are backported to the affected LTS branches
  - Then all product developers must determine if the bug and bug fix impact systems' safety of each system

MAINTENANCE ACTIVITY: ROOT CAUSE ANALYSIS

- **Rationale** for doing root-cause analysis during operations
  - Safety standard requires continuous monitoring and analysis of identified issues
- **Implementation** with the Linux kernel development
  - Bugs are continuously found in Linux
  - Bug-fix commits are backported to the affected LTS branches
  - Then all product developers must determine if the bug and bug fix impact systems' safety of each system
- **Process** for each bug-fix:
  - **Step 1** (once for each bug-fix):
    - **Kernel analysis team** describes the impact of a kernel bug on user space and its bug-fix in high detail
    - Analysis independent of the specific system, **one collaborative team**
  - **Step 2** (for each bug-fix and each system):
    - **System analysis team** judges if the described bug-fix impacts the system and the system's safety.
    - One team for each system employing Linux

CONCLUSION

MORE INFORMATION

- To use Linux properly in your safety-critical product, **join the SIL2LinuxMP Safety-Critical Linux working group**
- To join SIL2LinuxMP, contact Nicholas McGuire at safety-at-osadl.org

- **Upcoming Events**:
  - SIL2LinuxMP project management meeting, November 8th, 2017 at BMW Car IT in Munich, Germany
  - Seminar on IEC 61508 basics: Introduction to functional safety
    - useful for companies who want to join SIL2LinuxMP at this later stage
    - November 15th to 17th, 2017 at Virtual Vehicle in Graz, Austria
  - SIL2LinuxMP Workshop on Applying Linux Quality Assurance Methods
    - hands-on work with methods to assess & improve quality in the Linux kernel
    - December 5th to 7th, 2017 at BMW Car IT in Munich, Germany

- More literature & pointers to detailed slides & talks are in the references (last slide of this slide set)

CONCLUSION

– Multi-threaded, high-performant complex safety applications require qualification of a full-fledged operating system.
– SIL2LinuxMP project shows **feasibility of a Linux kernel safety qualification**.

– **The difference** between Safety-Critical Linux and main-line Linux **is the way you use it**.
  – Find evidences for OS functionalities and design your system to use those
  – Linux is not a generic Safety Element without system context (no SEooC!)
  – Arguments and evidences for the kernel's qualities can be created and maintained collaboratively.

– Safety-Critical Linux requires **quality assurance activities** with main-line accepted involvement.
– The Safety-Critical Linux Working Group brings **two groups** together:
  – **Product developers** & development companies of safety-critical systems
  – **Kernel developers** with interest in quality assurance
– If you are one of them, you are welcome to join the collaborative effort.

CONCLUSION

– Multi-threaded, high-performant complex safety applications require qualification of a full-fledged operating system.
– SIL2LinuxMP project shows **feasibility of a Linux kernel safety qualification.**

– **The difference** between Safety-Critical Linux and main-line Linux **is the way you use it**.
  – Find evidences for OS functionalities and design your system to use those
  – Linux is not a generic Safety Element without system context (no SEooC!)
  – Arguments and evidences for the kernel's qualities can be created and maintained collaboratively.

– Safety-Critical Linux requires **quality assurance activities** with main-line accepted involvement.
– The Safety-Critical Linux Working Group brings **two groups** together:
  – **Product developers** & development companies of safety-critical systems
  – **Kernel developers** with interest in quality assurance
– If you are one of them, you are welcome to join the collaborative effort.

Thanks for your attention!

REFERENCES

– OSADL, OSADL Project: SIL2LinuxMP, 2015. http://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html
– Nicholas McGuire, GNU/Linux for Safety Related Systems - Architectural and Procedural Issues. FOSDEM 2016. https://archive.fosdem.org/2016/schedule/event/safetysystems/
– Nicholas McGuire, Discussion of Statistical Methods for SIL2LinuxMP. Automotive Linux Summit 2016. http://www.opentech.at/slides/ALS_2016.pdf
– Andreas Platschek. Analyzing the Software Development Life-Cycle using Data-Mining Techniques. FOSDEM 2017. https://archive.fosdem.org/2017/schedule/event/software_life_cycle_data_mining/
– Geet Tapan Telang, Kotaro Hashimoto, Krishnaji Desai. Effective Source Code Analysis with Minimization. In: Operating Systems Platforms for Embedded Real-Time applications 2016. https://www.cs.hs-rm.de/~kaiser/events/ospert16/pdf/ospert16-p2.pdf
– Darren Hart, Would You Trust Linux with Your Life? Linux for Safety Critical Applications. Embedded Linux Conference 2016. https://www.youtube.com/watch?v=MSK6Aj6JsOc
– Wikipedia contributors, Functional safety. In: Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Functional_safety&oldid=796723717 (accessed October 19, 2017)

COPYRIGHT ON THIS PRESENTATION

Embedded Linux Conference

Europe

THE LINUX FOUNDATION